

1. This question involves the `Account` class, which is used to represent user accounts for a website. The `Account` class contains a helper method, `isAvailable`, which determines whether a username is available. You will write a constructor and a method in the `Account` class.

```
public class Account
{
    private String username; // To be initialized in part (a)

    /**
     * Creates a username based on the parameter requestedName. If the
     * username is unavailable, appends successive integers, beginning
     * with 1, to requestedName until an available username is found,
     * as described in part (a).
     */
    public Account(String requestedName)
    { /* to be implemented in part (a) */ }

    /**
     * Returns true if the parameter str is an available username;
     * returns false otherwise.
     */
    public static boolean isAvailable(String str)
    { /* implementation not shown */ }

    /**
     * Returns a shortened version of username with each hyphen ("-")
     * and the character before it removed, as described in part (b)
     * Preconditions: username does not start or end with a hyphen.
     *                  username does not contain consecutive hyphens.
     *                  username.length() >= 2
     * Postcondition: username is unchanged.
     */
    public String getShortenedName()
    { /* to be implemented in part (b) */ }

    /* There may be instance variables, constructors, and methods
       that are not shown. */
}
```

The following information applies to part A.

In part A, you will write the `Account` constructor, which creates a username based on the parameter `requestedName`.

A helper method, `isAvailable`, has been provided to determine whether a username is available. The `isAvailable` method returns `true` if its parameter is an available username and returns `false` otherwise.

If `requestedName` is an available username, the `Account` constructor will assign `requestedName` to the instance variable `username`. If not, the constructor will try different variations of `requestedName` until an available username is found and assigned to the instance variable `username`. The variations consist of `requestedName` followed by an integer, as in the following example.

- If `requestedName` is "Luis-Cruz" and this username is not available, then the constructor will check the availability of "Luis-Cruz1", "Luis-Cruz2", "Luis-Cruz3", and so on, until an available username is found. The first available username found is assigned to `username`.
- If `requestedName` is "PSmith" and this username is available, then "PSmith" is assigned to `username`.

The following information applies to part B.

In part B, you will write the `getShortenedName` method, which returns a shortened version of the instance variable `username` with each hyphen ("-") and the character immediately before it removed. If no hyphens appear in `username`, the value of `username` is returned.

For example, if `username` is "Amy-Marie-Lin", `getShortenedName` should return "AmMariLin".

As another example, if `username` is "SammyB3", `getShortenedName` should return "SammyB3".

Part A

Complete the `Account` constructor. You must use `isAvailable` appropriately in order to receive full credit.

```
/**
 * Creates a username based on the parameter requestedName. If the
 * username is unavailable, appends successive integers, beginning
 * with 1, to requestedName until an available username is found,
 * as described in part (a).
 */
public Account(String requestedName)
```

Part B

Complete method `getShortenedName`.

```
/**
 * Returns a shortened version of username with each hyphen ("-")
 * and the character before it removed, as described in part (b)
 * Preconditions: username does not start or end with a hyphen.
 *                username does not contain consecutive hyphens.
 *                username.length() >= 2
 * Postcondition: username is unchanged.
 */
public String getShortenedName()
```

3. This question involves the manipulation and analysis of a list of words. The following `WordChecker` class contains an `ArrayList<String>` to be analyzed and methods that are used to perform the analysis. You will write two methods of the `WordChecker` class.

```
public class WordChecker
{
    /** Initialized in the constructor and contains no null elements */
    private ArrayList<String> wordList;

    /**
     * Returns true if each element of wordList (except the first) contains the previous
     * element as a substring and returns false otherwise, as described in part (a)
     * Precondition: wordList contains at least two elements.
     * Postcondition: wordList is unchanged.
     */
    public boolean isWordChain()
    { /* to be implemented in part (a) */ }

    /**
     * Returns an ArrayList<String> based on strings from wordList that start
     * with target, as described in part (b). Each element of the returned ArrayList has had
     * the initial occurrence of target removed.
     * Postconditions: wordList is unchanged.
     * Items appear in the returned list in the same order as they appear in wordList.
     */
    public ArrayList<String> createList(String target)
    { /* to be implemented in part (b) */ }

    // There may be instance variables, constructors, and methods that are not shown.
}
```

- (a) Write the `isWordChain` method, which determines whether each element of `wordList` (except the first) contains the previous element as a substring. The following table shows two sample `isWordChain` method calls.

<code>wordList</code>	<code>isWordChain</code> Return Value	Explanation
<code>["an", "band", "band", "abandon"]</code>	<code>true</code>	Each element contains the previous element as a substring.
<code>["to", "too", "stool", "tools"]</code>	<code>false</code>	"tools" does not contain the substring "stool".

Complete the `isWordChain` method.

```
/**
 * Returns true if each element of wordList (except the first) contains the previous
 * element as a substring and returns false otherwise, as described in part (a)
 * Precondition: wordList contains at least two elements.
 * Postcondition: wordList is unchanged.
 */
public boolean isWordChain()
```

Begin your response at the top of a new page in the separate Free Response booklet and fill in the appropriate circle at the top of each page to indicate the question number. If there are multiple parts to this question, write the part letter with your response.

- (b) Write the `createList` method, which creates and returns an `ArrayList<String>`. The method identifies strings in `wordList` that start with `target` and returns a new `ArrayList` containing each identified string without the starting occurrence of `target`. Elements must appear in the returned list in the same order as they appear in `wordList`.

Consider an example where `wordList` contains the following strings.

```
["catch", "bobcat", "catchacat", "cat", "at"]
```

The following table shows the `ArrayList` returned by some calls to `createList`. In all cases, `wordList` is unchanged.

Method Call	ArrayList Returned by <code>createList</code>	Explanation
<code>createList("cat")</code>	<code>["ch", "chacat", ""]</code>	Only "catch", "catchacat", and "cat" begin with "cat".
<code>createList("catch")</code>	<code>["", "acat"]</code>	Only "catch" and "catchacat" begin with "catch".
<code>createList("dog")</code>	<code>[]</code>	None of the words in <code>wordList</code> begin with "dog".

Complete the `createList` method.

```
/**
 * Returns an ArrayList<String> based on strings from wordList that start
 * with target, as described in part (b). Each element of the returned ArrayList has had
 * the initial occurrence of target removed.
 * Postconditions: wordList is unchanged.
 * Items appear in the returned list in the same order as they appear in wordList.
 */
public ArrayList<String> createList(String target)
```

Begin your response at the top of a new page in the separate Free Response booklet and fill in the appropriate circle at the top of each page to indicate the question number. If there are multiple parts to this question, write the part letter with your response.

Class information for this question

```
public class WordChecker
private ArrayList<String> wordList
public boolean isWordChain()
public ArrayList<String> createList(String target)
```

3. Users of a website are asked to provide a review of the website at the end of each visit. Each review, represented by an object of the `Review` class, consists of an integer indicating the user's rating of the website and an optional `String` comment field. The comment field in a `Review` object ends with a period (". "), exclamation point ("! "), or letter, or is a `String` of length 0 if the user did not enter a comment.

```
public class Review
{
    private int rating;
    private String comment;

    /** Precondition: r >= 0
     *      c is not null.
     */
    public Review(int r, String c)
    {
        rating = r;
        comment = c;
    }

    public int getRating()
    {
        return rating;
    }

    public String getComment()
    {
        return comment;
    }

    // There may be instance variables, constructors, and methods that are not shown.
}
```

GO ON TO THE NEXT PAGE.

The `ReviewAnalysis` class contains methods used to analyze the reviews provided by users. You will write two methods of the `ReviewAnalysis` class.

```
public class ReviewAnalysis
{
    /** All user reviews to be included in this analysis */
    private Review[] allReviews;

    /** Initializes allReviews to contain all the Review objects to be analyzed */
    public ReviewAnalysis()
    { /* implementation not shown */ }

    /** Returns a double representing the average rating of all the Review objects to be
     * analyzed, as described in part (a)
     * Precondition: allReviews contains at least one Review.
     * No element of allReviews is null.
     */
    public double getAverageRating()
    { /* to be implemented in part (a) */ }

    /** Returns an ArrayList of String objects containing formatted versions of
     * selected user comments, as described in part (b)
     * Precondition: allReviews contains at least one Review.
     * No element of allReviews is null.
     * Postcondition: allReviews is unchanged.
     */
    public ArrayList<String> collectComments()
    { /* to be implemented in part (b) */ }
}
```

GO ON TO THE NEXT PAGE.

- (a) Write the `ReviewAnalysis` method `getAverageRating`, which returns the average rating (arithmetic mean) of all elements of `allReviews`. For example, `getAverageRating` would return 3.4 if `allReviews` contained the following `Review` objects.

0	1	2	3	4
4 "Good! Thx"	3 "OK site"	5 "Great!"	2 "Poor! Bad."	3 ""

Complete method `getAverageRating`.

```
/** Returns a double representing the average rating of all the Review objects to be
 * analyzed, as described in part (a)
 * Precondition: allReviews contains at least one Review.
 * No element of allReviews is null.
 */
public double getAverageRating()
```

**Begin your response at the top of a new page in the Free Response booklet
and fill in the appropriate circle indicating the question number.
If there are multiple parts to this question, write the part letter with your response.**

Class information for this question

```
public class Review
private int rating
private String comment
public Review(int r, String c)
public int getRating()
public String getComment()
public class ReviewAnalysis
private Review[] allReviews
public ReviewAnalysis()
public double getAverageRating()
public ArrayList<String> collectComments()
```

GO ON TO THE NEXT PAGE.

(b) Write the `ReviewAnalysis` method `collectComments`, which collects and formats only comments that contain an exclamation point. The method returns an `ArrayList` of `String` objects containing copies of user comments from `allReviews` that contain an exclamation point, formatted as follows. An empty `ArrayList` is returned if no comment in `allReviews` contains an exclamation point.

- The `String` inserted into the `ArrayList` to be returned begins with the index of the `Review` in `allReviews`.
- The index is immediately followed by a hyphen (" - ").
- The hyphen is followed by a copy of the original comment.
- The `String` must end with either a period or an exclamation point. If the original comment from `allReviews` does not end in either a period or an exclamation point, a period is added.

The following example of `allReviews` is repeated from part (a).

0	1	2	3	4
4 "Good! Thx"	3 "OK site"	5 "Great!"	2 "Poor! Bad."	3 ""

The following `ArrayList` would be returned by a call to `collectComments` with the given contents of `allReviews`. The reviews at index 1 and index 4 in `allReviews` are not included in the `ArrayList` to return since neither review contains an exclamation point.

"0-Good! Thx."	"2-Great!"	"3-Poor! Bad."
----------------	------------	----------------

Complete method `collectComments`.

```
/** Returns an ArrayList of String objects containing formatted versions of
 * selected user comments, as described in part (b)
 * Precondition: allReviews contains at least one Review.
 * No element of allReviews is null.
 * Postcondition: allReviews is unchanged.
 */
public ArrayList<String> collectComments()
```

Begin your response at the top of a new page in the Free Response booklet and fill in the appropriate circle indicating the question number. If there are multiple parts to this question, write the part letter with your response.

GO ON TO THE NEXT PAGE.

1. This question involves the `WordMatch` class, which stores a secret string and provides methods that compare other strings to the secret string. You will write two methods in the `WordMatch` class.

```
public class WordMatch
{
    /** The secret string. */
    private String secret;

    /** Constructs a WordMatch object with the given secret string of lowercase letters. */
    public WordMatch(String word)
    {
        /* implementation not shown */
    }

    /** Returns a score for guess, as described in part (a).
     * Precondition: 0 < guess.length() <= secret.length()
     */
    public int scoreGuess(String guess)
    { /* to be implemented in part (a) */ }

    /** Returns the better of two guesses, as determined by scoreGuess and the rules for a
     * tie-breaker that are described in part (b).
     * Precondition: guess1 and guess2 contain all lowercase letters.
     * guess1 is not the same as guess2.
     */
    public String findBetterGuess(String guess1, String guess2)
    { /* to be implemented in part (b) */ }
}
```

- (a) Write the `WordMatch` method `scoreGuess`. To determine the score to be returned, `scoreGuess` finds the number of times that `guess` occurs as a substring of `secret` and then multiplies that number by the square of the length of `guess`. Occurrences of `guess` may overlap within `secret`.

Assume that the length of `guess` is less than or equal to the length of `secret` and that `guess` is not an empty string.

The following examples show declarations of a `WordMatch` object. The tables show the outcomes of some possible calls to the `scoreGuess` method.

```
WordMatch game = new WordMatch("mississippi");
```

Value of <code>guess</code>	Number of Substring Occurrences	Score Calculation: (Number of Substring Occurrences) x (Square of the Length of <code>guess</code>)	Return Value of <code>game.scoreGuess(guess)</code>
"i"	4	$4 * 1 * 1 = 4$	4
"iss"	2	$2 * 3 * 3 = 18$	18
"issipp"	1	$1 * 6 * 6 = 36$	36
"mississippi"	1	$1 * 11 * 11 = 121$	121

```
WordMatch game = new WordMatch("aaaabb");
```

Value of <code>guess</code>	Number of Substring Occurrences	Score Calculation: (Number of Substring Occurrences) x (Square of the Length of <code>guess</code>)	Return Value of <code>game.scoreGuess(guess)</code>
"a"	4	$4 * 1 * 1 = 4$	4
"aa"	3	$3 * 2 * 2 = 12$	12
"aaa"	2	$2 * 3 * 3 = 18$	18
"aabb"	1	$1 * 4 * 4 = 16$	16
"c"	0	$0 * 1 * 1 = 0$	0

Complete the `scoreGuess` method.

```
/** Returns a score for guess, as described in part (a).  
 * Precondition: 0 < guess.length() <= secret.length()  
 */  
public int scoreGuess(String guess)
```

Begin your response at the top of a new page in the separate Free Response booklet and fill in the appropriate circle at the top of each page to indicate the question number. If there are multiple parts to this question, write the part letter with your response.

Class information for this question

```
public class WordMatch
```

```
private String secret
```

```
public WordMatch(String word)
```

```
public int scoreGuess(String guess)
```

```
public String findBetterGuess(String guess1, String guess2)
```

- (b) Write the `WordMatch` method `findBetterGuess`, which returns the better guess of its two `String` parameters, `guess1` and `guess2`. If the `scoreGuess` method returns different values for `guess1` and `guess2`, then the guess with the higher score is returned. If the `scoreGuess` method returns the same value for `guess1` and `guess2`, then the alphabetically greater guess is returned.

The following example shows a declaration of a `WordMatch` object and the outcomes of some possible calls to the `scoreGuess` and `findBetterGuess` methods.

```
WordMatch game = new WordMatch("concatenation");
```

Method Call	Return Value	Explanation
<code>game.scoreGuess("ten");</code>	9	1 * 3 * 3
<code>game.scoreGuess("nation");</code>	36	1 * 6 * 6
<code>game.findBetterGuess("ten", "nation");</code>	"nation"	Since <code>scoreGuess</code> returns 36 for "nation" and 9 for "ten", the guess with the greater score, "nation", is returned.
<code>game.scoreGuess("con");</code>	9	1 * 3 * 3
<code>game.scoreGuess("cat");</code>	9	1 * 3 * 3
<code>game.findBetterGuess("con", "cat");</code>	"con"	Since <code>scoreGuess</code> returns 9 for both "con" and "cat", the alphabetically greater guess, "con", is returned.

Complete method `findBetterGuess`.

Assume that `scoreGuess` works as specified, regardless of what you wrote in part (a). You must use `scoreGuess` appropriately to receive full credit.

```
/** Returns the better of two guesses, as determined by scoreGuess and the rules for a
 * tie-breaker that are described in part (b).
 * Precondition: guess1 and guess2 contain all lowercase letters.
 * guess1 is not the same as guess2.
 */
public String findBetterGuess(String guess1, String guess2)
```

Begin your response at the top of a new page in the separate Free Response booklet and fill in the appropriate circle at the top of each page to indicate the question number. If there are multiple parts to this question, write the part letter with your response.

Class information for this question

```
public class WordMatch
```

```
private String secret
```

```
public WordMatch(String word)
```

```
public int scoreGuess(String guess)
```

```
public String findBetterGuess(String guess1, String guess2)
```

3. This question involves analyzing and modifying a string. The following `Phrase` class maintains a phrase in an instance variable and has methods that access and make changes to the phrase. You will write two methods of the `Phrase` class.

```
public class Phrase
{
    private String currentPhrase;

    /** Constructs a new Phrase object. */
    public Phrase(String p)
    {
        currentPhrase = p;
    }

    /** Returns the index of the nth occurrence of str in the current phrase;
     *  returns -1 if the nth occurrence does not exist.
     *  Precondition: str.length() > 0 and n > 0
     *  Postcondition: the current phrase is not modified.
     */
    public int findNthOccurrence(String str, int n)
    {
        /* implementation not shown */
    }

    /** Modifies the current phrase by replacing the nth occurrence of str with repl.
     *  If the nth occurrence does not exist, the current phrase is unchanged.
     *  Precondition: str.length() > 0 and n > 0
     */
    public void replaceNthOccurrence(String str, int n, String repl)
    {
        /* to be implemented in part (a) */
    }

    /** Returns the index of the last occurrence of str in the current phrase;
     *  returns -1 if str is not found.
     *  Precondition: str.length() > 0
     *  Postcondition: the current phrase is not modified.
     */
    public int findLastOccurrence(String str)
    {
        /* to be implemented in part (b) */
    }

    /** Returns a string containing the current phrase. */
    public String toString()
    {
        return currentPhrase;
    }
}
```

Part (a) begins on page 12.

- (a) Write the Phrase method `replaceNthOccurrence`, which will replace the `nth` occurrence of the string `str` with the string `repl`. If the `nth` occurrence does not exist, `currentPhrase` remains unchanged.

Several examples of the behavior of the method `replaceNthOccurrence` are shown below.

Code segments	Output produced
<pre>Phrase phrase1 = new Phrase("A cat ate late."); phrase1.replaceNthOccurrence("at", 1, "rane"); System.out.println(phrase1);</pre>	A crane ate late.
<pre>Phrase phrase2 = new Phrase("A cat ate late."); phrase2.replaceNthOccurrence("at", 6, "xx"); System.out.println(phrase2);</pre>	A cat ate late.
<pre>Phrase phrase3 = new Phrase("A cat ate late."); phrase3.replaceNthOccurrence("bat", 2, "xx"); System.out.println(phrase3);</pre>	A cat ate late.
<pre>Phrase phrase4 = new Phrase("aaaa"); phrase4.replaceNthOccurrence("aa", 1, "xx"); System.out.println(phrase4);</pre>	xxaa
<pre>Phrase phrase5 = new Phrase("aaaa"); phrase5.replaceNthOccurrence("aa", 2, "bbb"); System.out.println(phrase5);</pre>	abbba

Class information for this question

```
public class Phrase
```

```
private String currentPhrase
```

```
public Phrase(String p)
```

```
public int findNthOccurrence(String str, int n)
```

```
public void replaceNthOccurrence(String str, int n, String repl)
```

```
public int findLastOccurrence(String str)
```

```
public String toString()
```

The `Phrase` class includes the method `findNthOccurrence`, which returns the `nth` occurrence of a given string. You must use `findNthOccurrence` appropriately to receive full credit.

Complete method `replaceNthOccurrence` below.

```
/** Modifies the current phrase by replacing the nth occurrence of str with repl.
 * If the nth occurrence does not exist, the current phrase is unchanged.
 * Precondition: str.length() > 0 and n > 0
 */
public void replaceNthOccurrence(String str, int n, String repl)
```

Part (b) begins on page 14.

- (b) Write the `Phrase` method `findLastOccurrence`. This method finds and returns the index of the last occurrence of a given string in `currentPhrase`. If the given string is not found, `-1` is returned. The following tables show several examples of the behavior of the method `findLastOccurrence`.

```
Phrase phrase1 = new Phrase("A cat ate late.");
```

Method call	Value returned
<code>phrase1.findLastOccurrence("at")</code>	11
<code>phrase1.findLastOccurrence("cat")</code>	2
<code>phrase1.findLastOccurrence("bat")</code>	-1

Class information for this question

```
public class Phrase
```

```
private String currentPhrase
public Phrase(String p)
public int findNthOccurrence(String str, int n)
public void replaceNthOccurrence(String str, int n, String repl)
public int findLastOccurrence(String str)
public String toString()
```

WRITE YOUR SOLUTION ON THE NEXT PAGE.

You must use `findNthOccurrence` appropriately to receive full credit.

Complete method `findLastOccurrence` below.

```
/** Returns the index of the last occurrence of str in the current phrase;
 * returns -1 if str is not found.
 * Precondition: str.length() > 0
 * Postcondition: the current phrase is not modified.
 */
```

2. This question involves two classes that are used to process log messages. A list of sample log messages is given below.

```
CLIENT3:security alert - repeated login failures
Webserver:disk offline
SERVER1:file not found
SERVER2:read error on disk DSK1
SERVER1:write error on disk DSK2
Webserver:error on /dev/disk
```

Log messages have the format *machineId:description*, where *machineId* identifies the computer and *description* describes the event being logged. Exactly one colon (":") appears in a log message. There are no blanks either immediately before or immediately after the colon.

The following `LogMessage` class is used to represent a log message.

```
public class LogMessage
{
    private String machineId;
    private String description;

    /** Precondition: message is a valid log message. */
    public LogMessage(String message)
    { /* to be implemented in part (a) */ }

    /** Returns true if the description in this log message properly contains keyword;
     *     false otherwise.
     */
    public boolean containsWord(String keyword)
    { /* to be implemented in part (b) */ }

    public String getMachineId()
    { return machineId; }

    public String getDescription()
    { return description; }

    // There may be instance variables, constructors, and methods that are not shown.
}
```

- (a) Write the constructor for the `LogMessage` class. It must initialize the private data of the object so that `getMachineId` returns the *machineId* part of the message and `getDescription` returns the *description* part of the message.

Complete the `LogMessage` constructor below.

```
/** Precondition: message is a valid log message. */  
public LogMessage(String message)
```

Part (b) begins on page 8.

- (b) Write the `LogMessage` method `containsWord`, which returns `true` if the description in the log message *properly contains* a given keyword and returns `false` otherwise.

A description *properly contains* a keyword if all three of the following conditions are true.

- the keyword is a substring of the description;
- the keyword is either at the beginning of the description or it is immediately preceded by a space;
- the keyword is either at the end of the description or it is immediately followed by a space.

The following tables show several examples. The descriptions in the left table properly contain the keyword `"disk"`. The descriptions in the right table do not properly contain the keyword `"disk"`.

Descriptions that properly contain `"disk"`

<code>"disk"</code>
<code>"error on disk"</code>
<code>"error on /dev/disk disk"</code>
<code>"error on disk DSK1"</code>

Descriptions that do not properly contain `"disk"`

<code>"DISK"</code>
<code>"error on disk3"</code>
<code>"error on /dev/disk"</code>
<code>"diskette"</code>

WRITE YOUR SOLUTION ON THE NEXT PAGE.

Assume that the `LogMessage` constructor works as specified, regardless of what you wrote in part (a).

Complete method `containsWord` below.

```
/** Returns true if the description in this log message properly contains keyword;  
 *      false otherwise.  
 */  
public boolean containsWord(String keyword)
```

Part (c) begins on page 10.

- (c) The `SystemLog` class represents a list of `LogMessage` objects and provides a method that removes and returns a list of all log messages (if any) that properly contain a given keyword. The messages in the returned list appear in the same order in which they originally appeared in the system log. If no message properly contains the keyword, an empty list is returned. The declaration of the `SystemLog` class is shown below.

```
public class SystemLog
{
    /** Contains all the entries in this system log.
     *  Guaranteed not to be null and to contain only non-null entries.
     */
    private List<LogMessage> messageList;

    /** Removes from the system log all entries whose descriptions properly contain keyword,
     *  and returns a list (possibly empty) containing the removed entries.
     *  Postcondition:
     *  - Entries in the returned list properly contain keyword and
     *    are in the order in which they appeared in the system log.
     *  - The remaining entries in the system log do not properly contain keyword and
     *    are in their original order.
     *  - The returned list is empty if no messages properly contain keyword.
     */
    public List<LogMessage> removeMessages(String keyword)
    { /* to be implemented in part (c) */ }

    // There may be instance variables, constructors, and methods that are not shown.
}
```

Write the `SystemLog` method `removeMessages`, which removes from the system log all entries whose descriptions properly contain `keyword` and returns a list of the removed entries in their original order. For example, assume that `theLog` is a `SystemLog` object initially containing six `LogMessage` objects representing the following list of log messages.

```
CLIENT3:security alert - repeated login failures
Webserver:disk offline
SERVER1:file not found
SERVER2:read error on disk DSK1
SERVER1:write error on disk DSK2
Webserver:error on /dev/disk
```

The call `theLog.removeMessages("disk")` would return a list containing the `LogMessage` objects representing the following log messages.

```
Webserver:disk offline
SERVER2:read error on disk DSK1
SERVER1:write error on disk DSK2
```

After the call, `theLog` would contain the following log messages.

```
CLIENT3:security alert - repeated login failures
SERVER1:file not found
Webserver:error on /dev/disk
```

Assume that the `LogMessage` class works as specified, regardless of what you wrote in parts (a) and (b). You must use `containsWord` appropriately to receive full credit.

Complete method `removeMessages` below.

```
/** Removes from the system log all entries whose descriptions properly contain keyword,
 * and returns a list (possibly empty) containing the removed entries.
 * Postcondition:
 *   - Entries in the returned list properly contain keyword and
 *     are in the order in which they appeared in the system log.
 *   - The remaining entries in the system log do not properly contain keyword and
 *     are in their original order.
 *   - The returned list is empty if no messages properly contain keyword.
 */
public List<LogMessage> removeMessages(String keyword)
```

COMPUTER SCIENCE A**SECTION II****Time—1 hour and 45 minutes****Number of questions—4****Percent of total score—50**

Directions: **SHOW ALL YOUR WORK. REMEMBER THAT PROGRAM SEGMENTS ARE TO BE WRITTEN IN JAVA.**

Notes:

- Assume that the classes listed in the Java Quick Reference have been imported where appropriate.
 - Unless otherwise noted in the question, assume that parameters in method calls are not `null` and that methods are called only when their preconditions are satisfied.
 - In writing solutions for each question, you may use any of the accessible methods that are listed in classes defined in that question. Writing significant amounts of code that can be replaced by a call to one of these methods will not receive full credit.
1. This question involves reasoning about one-dimensional and two-dimensional arrays of integers. You will write three static methods, all of which are in a single enclosing class, named `DiverseArray` (not shown). The first method returns the sum of the values of a one-dimensional array; the second method returns an array that represents the sums of the rows of a two-dimensional array; and the third method analyzes row sums.
- (a) Write a static method `arraySum` that calculates and returns the sum of the entries in a specified one-dimensional array. The following example shows an array `arr1` and the value returned by a call to `arraySum`.

<u>arr1</u>					Value returned by <u>arraySum(arr1)</u>
0	1	2	3	4	
1	3	2	7	3	16

WRITE YOUR SOLUTION ON THE NEXT PAGE.

Complete method `arraySum` below.

```
/** Returns the sum of the entries in the one-dimensional array arr.  
 */  
public static int arraySum(int[] arr)
```

Part (b) begins on page 4.

- (b) Write a static method `rowSums` that calculates the sums of each of the rows in a given two-dimensional array and returns these sums in a one-dimensional array. The method has one parameter, a two-dimensional array `arr2D` of `int` values. The array is in row-major order: `arr2D[r][c]` is the entry at row `r` and column `c`. The method returns a one-dimensional array with one entry for each row of `arr2D` such that each entry is the sum of the corresponding row in `arr2D`. As a reminder, each row of a two-dimensional array is a one-dimensional array.

For example, if `mat1` is the array represented by the following table, the call `rowSums(mat1)` returns the array `{16, 32, 28, 20}`.

	<u>mat1</u>				
	0	1	2	3	4
0	1	3	2	7	3
1	10	10	4	6	2
2	5	3	5	9	6
3	7	6	4	2	1

Methods written in this question

```
public static int arraySum(int[] arr)
public static int[] rowSums(int[][] arr2D)
public static boolean isDiverse(int[][] arr2D)
```

WRITE YOUR SOLUTION ON THE NEXT PAGE.

Assume that `arraySum` works as specified, regardless of what you wrote in part (a). You must use `arraySum` appropriately to receive full credit.

Complete method `rowSums` below.

```
/** Returns a one-dimensional array in which the entry at index k is the sum of
 * the entries of row k of the two-dimensional array arr2D.
 */
public static int[] rowSums(int[][] arr2D)
```

Part (c) begins on page 6.

- (c) A two-dimensional array is *diverse* if no two of its rows have entries that sum to the same value. In the following examples, the array `mat1` is diverse because each row sum is different, but the array `mat2` is not diverse because the first and last rows have the same sum.

	<u>mat1</u>					
	0	1	2	3	4	Row sums
0	1	3	2	7	3	16
1	10	10	4	6	2	32
2	5	3	5	9	6	28
3	7	6	4	2	1	20

	<u>mat2</u>					
	0	1	2	3	4	Row sums
0	1	1	5	3	4	14
1	12	7	6	1	9	35
2	8	11	10	2	5	36
3	3	2	3	0	6	14

Write a static method `isDiverse` that determines whether or not a given two-dimensional array is diverse. The method has one parameter: a two-dimensional array `arr2D` of `int` values. The method should return `true` if all the row sums in the given array are unique; otherwise, it should return `false`. In the arrays shown above, the call `isDiverse(mat1)` returns `true` and the call `isDiverse(mat2)` returns `false`.

Methods written in this question

```
public static int arraySum(int[] arr)
public static int[] rowSums(int[][] arr2D)
public static boolean isDiverse(int[][] arr2D)
```

Assume that `arraySum` and `rowSums` work as specified, regardless of what you wrote in parts (a) and (b). You must use `rowSums` appropriately to receive full credit.

Complete method `isDiverse` below.

```
/** Returns true if all rows in arr2D have different row sums;
 *     false otherwise.
 */
public static boolean isDiverse(int[][] arr2D)
```

2. Consider a guessing game in which a player tries to guess a hidden word. The hidden word contains only capital letters and has a length known to the player. A guess contains only capital letters and has the same length as the hidden word.

After a guess is made, the player is given a hint that is based on a comparison between the hidden word and the guess. Each position in the hint contains a character that corresponds to the letter in the same position in the guess. The following rules determine the characters that appear in the hint.

If the letter in the guess is ...

the corresponding character in the hint is

also in the same position in the hidden word,	the matching letter
also in the hidden word, but in a different position,	" + "
not in the hidden word,	" * "

The `HiddenWord` class will be used to represent the hidden word in the game. The hidden word is passed to the constructor. The class contains a method, `getHint`, that takes a guess and produces a hint.

For example, suppose the variable `puzzle` is declared as follows.

```
HiddenWord puzzle = new HiddenWord("HARPS");
```

The following table shows several guesses and the hints that would be produced.

Call to <code>getHint</code>	String returned
<code>puzzle.getHint("AAAAA")</code>	" +A+++ "
<code>puzzle.getHint("HELLO")</code>	"H**** "
<code>puzzle.getHint("HEART")</code>	"H*++* "
<code>puzzle.getHint("HARMS")</code>	"HAR*S "
<code>puzzle.getHint("HARPS")</code>	"HARPS "

Write the complete `HiddenWord` class, including any necessary instance variables, its constructor, and the method, `getHint`, described above. You may assume that the length of the guess is the same as the length of the hidden word.

COMPUTER SCIENCE A**SECTION II****Time—1 hour and 45 minutes****Number of questions—4****Percent of total score—50**

Directions: SHOW ALL YOUR WORK. REMEMBER THAT PROGRAM SEGMENTS ARE TO BE WRITTEN IN JAVA.

Notes:

- Assume that the classes listed in the appendices have been imported where appropriate.
 - Unless otherwise noted in the question, assume that parameters in method calls are not `null` and that methods are called only when their preconditions are satisfied.
 - In writing solutions for each question, you may use any of the accessible methods that are listed in classes defined in that question. Writing significant amounts of code that can be replaced by a call to one of these methods may not receive full credit.
1. This question involves reasoning about strings made up of uppercase letters. You will implement two related methods that appear in the same class (not shown). The first method takes a single string parameter and returns a scrambled version of that string. The second method takes a list of strings and modifies the list by scrambling each entry in the list. Any entry that cannot be scrambled is removed from the list.
- (a) Write the method `scrambleWord`, which takes a given word and returns a string that contains a scrambled version of the word according to the following rules.
- The scrambling process begins at the first letter of the word and continues from left to right.
 - If two consecutive letters consist of an "A" followed by a letter that is not an "A", then the two letters are swapped in the resulting string.
 - Once the letters in two adjacent positions have been swapped, neither of those two positions can be involved in a future swap.

The following table shows several examples of words and their scrambled versions.

word	Result returned by <code>scrambleWord(word)</code>
"TAN"	"TNA"
"ABRACADABRA"	"BARCADABARA"
"WHOA"	"WHOA"
"AARDVARK"	"ARADVRAK"
"EGGS"	"EGGS"
"A"	"A"
""	""

WRITE YOUR SOLUTION ON THE NEXT PAGE.

Complete method `scrambleWord` below.

```
/** Scrambles a given word.
 * @param word the word to be scrambled
 * @return the scrambled word (possibly equal to word)
 * Precondition: word is either an empty string or contains only uppercase letters.
 * Postcondition: the string returned was created from word as follows:
 *   - the word was scrambled, beginning at the first letter and continuing from left to right
 *   - two consecutive letters consisting of "A" followed by a letter that was not "A" were swapped
 *   - letters were swapped at most once
 */
public static String scrambleWord(String word)
```

Part (b) begins on page 4.

- (b) Write the method `scrambleOrRemove`, which replaces each word in the parameter `wordList` with its scrambled version and removes any words that are unchanged after scrambling. The relative ordering of the entries in `wordList` remains the same as before the call to `scrambleOrRemove`.

The following example shows how the contents of `wordList` would be modified as a result of calling `scrambleOrRemove`.

Before the call to `scrambleOrRemove`:

	0	1	2	3	4
wordList	"TAN"	"ABRACADABRA"	"WHOA"	"APPLE"	"EGGS"

After the call to `scrambleOrRemove`:

	0	1	2
wordList	"TNA"	"BARCADABARA"	"PAPLE"

WRITE YOUR SOLUTION ON THE NEXT PAGE.

Assume that `scrambleWord` is in the same class as `scrambleOrRemove` and works as specified, regardless of what you wrote in part (a).

Complete method `scrambleOrRemove` below.

```
/** Modifies wordList by replacing each word with its scrambled
 * version, removing any words that are unchanged as a result of scrambling.
 * @param wordList the list of words
 * Precondition: wordList contains only non-null objects
 * Postcondition:
 *   - all words unchanged by scrambling have been removed from wordList
 *   - each of the remaining words has been replaced by its scrambled version
 *   - the relative ordering of the entries in wordList is the same as it was
 *     before the method was called
 */
public static void scrambleOrRemove(List<String> wordList)
```